
MOAB Tutorial

September 29, 2010

Outline

- ITAPS Data Model
- iMesh Interface (w/ examples)
- MOAB vs. ITAPS
- Best Practices (for Performance)
- Parallel Data

Introduction

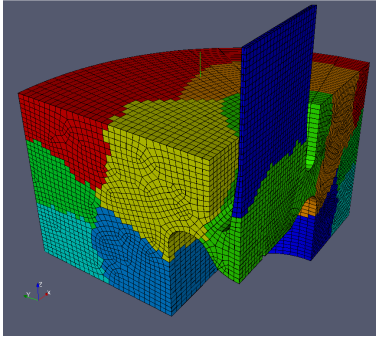
- MOAB:
 - A database for mesh (structured and unstructured) and field data associated with mesh
 - Tuned for memory efficiency first, speed a close second
 - C++ in both implementation and interface
 - Serial, parallel look very similar, parallel data constructs embedded in MOAB/iMesh data model
- ITAPS iMesh:
 - A common API, data model for accessing mesh, field data
 - C, directly-callable from C, C++, Fortran
 - Pytaps for accessing from Python
 - Numpy arrays for fine-grained data, efficient

ITAPS Data Model

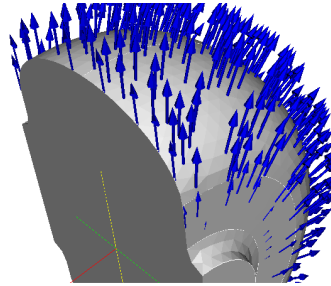
- 4 fundamental “types”:
 - *Entity*: fine-grained entities in interface (vertex, tri, hex)
 - Supported types: **vertex**, **edge**, **tri**, **quad**, **polygon**, **tet**, **prism**, **pyramid**, **hex**, **septahedron**, **polyhedron**
 - Mostly unstructured, though some implementations (MOAB) can represent structured & (soon) expose in ITAPS data model
 - Flexible in representing intermediate-dimension entities (internal edges/faces)
 - *Entity Set*: arbitrary set of entities & other sets
 - Parent/child relations, for embedded graphs between sets
 - *Interface*: object on which interface functions are called and through which other data are obtained
 - *Tag*: named datum annotated to Entitys, Entity Sets, Interface
- Instances accessed using opaque (type-less) “handles”

ITAPS Data Model Usage

Mesh Partition



Klystron mesh, SLAC/SNL

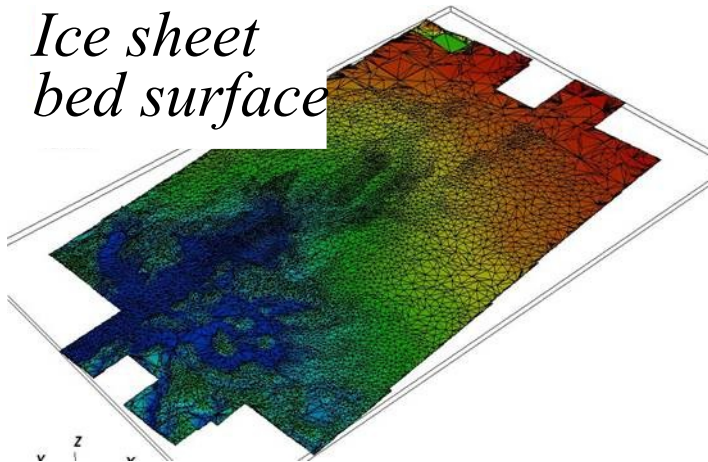


*Vertex-based
vector field*

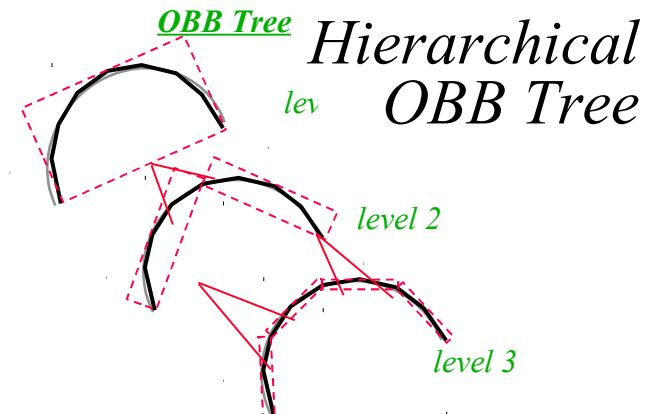
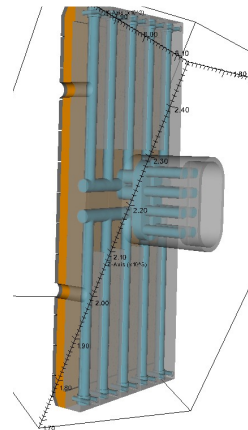


*Inside/outside on
structured mesh*

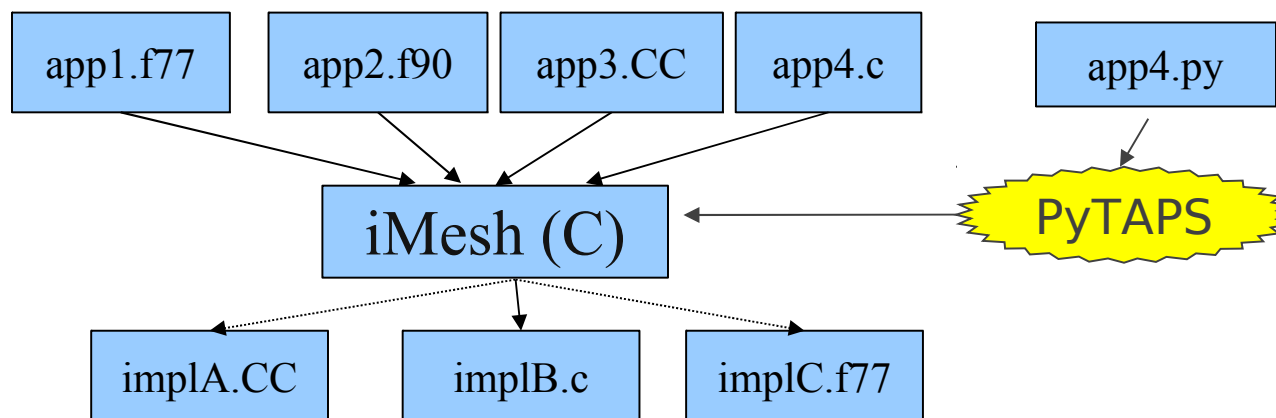
Ice sheet bed surface



Geom model facets, topology



ITAPS Interface Design



- C-based interface, but designed to be callable directly from Fortran and C++
 - Good portability, performance
 - Maintenance easier
 - iGeom, iRel too
- Quick startup for new users

Simple Example: HELLO iMesh (C++)

- Simple, typical application which 1) Instantiates iMesh interface, 2) Reads mesh from disk, 3) Reports # entities of each dimension

```
#include <iostream>
#include "iMesh.h"

int main( int argc, char *argv[] )
{
    // create the Mesh instance
    char *options = NULL;
    iMesh_Instance mesh;
    int ierr, options_len = 0;
    iMesh_newMesh(options, &mesh, &ierr,
                  options_len);
```

1

```
    // load the mesh
    iMesh_load(mesh, argv[1], options, &ierr,
               strlen(argv[1]), options_len);
```

2

```
    // report the number of elements of each dimension
    for (int dim = iBase_VERTEX; dim <= iBase_REGION; dim++) {
        int numd;
        iMesh_getNumOfType(mesh, 0, dim, &numd, &ierr);
        std::cout << "Number of " << dim << "d elements = "
                  << numd << std::endl;
```

3

```
    }
    return true;}

```

Makefile:

```
include ../../iMesh-Defs.inc

HELLOiMesh: HELLOiMesh.o
    $(CXX) $(CXXFLAGS) -o $@ HELLOiMesh.o \
        ${IMESH_LIBS}

.cpp.o:
    ${CXX} -c ${CXXFLAGS} $IMESH_INCLUDES} $<
```

*Note: no error checking here for brevity,
but there should be in your code!!!*

ITAPS API's: Argument Handling Conventions

- ITAPS API's are C-like and can be called directly from C, Fortran, C++
- Arguments pass by value (in) or reference (inout, out)
 - Fortran: use %VAL extension
- Memory allocation for lists done in application *or* implementation
 - If inout list comes in allocated, length must be long enough to store results of call
 - By definition, allocation/deallocation done using C malloc/free; application required to free memory returned by implementation
 - Fortran: Use “cray pointer” extension (equivalences to normal f77 array)
- Handle types typedef'd to size_t (iBase_EntityHandle, iBase_EntitySetHandle, iBase_TagHandle, iMesh_Instance)
- Strings: char*, with length passed by value after all other args
- Enum's: *values (iBase_SUCCESS, etc.) available for comparison operations, but passed as integer arguments*
 - Fortran: *named parameters*

Argument Handling Conventions

Issue	C	FORTRAN
Function Names	iXxxx_ prefix	Same as C
Interface Handle	Typedef'd to size_t, as type iXxxx_Instance; instance handle is f argument to all functions	#define'd as type Integer; handle instance is i argument to all functions
Enumerated Variables	All arguments integer-type instead of enum-type; values from enumerated types	Same, with enum values defined as FORTRAN parameters
Entity, Set, Tag Handles	Typedef'd as size_t; typedef types iBase_EntityHandle, iBase_EntitySetHandle, iBase_TagHandle	#define'd as type Integer
Lists	<ul style="list-style-type: none"> • In: X *list, int occupied_size • Inout: X **list, int *allocated_size, int **occupied_size • malloc/free-based memory allocation/deallocation 	Same, with Cray pointers used to reference arrays (see FindConnectF example)
String	char*-type, with string length(s) at end of argument list	char[]-type without extra length argument (this length gets added implicitly by FORTRAN compiler)

ITAPS API

- Important enumerated types:
 - EntityType (iBase_VERTEX, EDGE, FACE, REGION)
 - EntityTopology (iMesh_POINT, LINE, TRI, QUAD, ...)
 - StorageOrder (iBase_BLOCKED, INTERLEAVED)
 - TagDataType (iBase_INTEGER, DOUBLE, ENTITY_HANDLE)
 - ErrorType (iBase_SUCCESS, iBase_FAILURE, ...)
- Enumerated type & function names have iBase, iMesh, iGeom, other names prepended

iMesh API Summary

- Basic (Mesh): load, save, getEntities, getNumOfType/Topo, getAllVtxCoordinates, getAdjacencies
- Entity: init/get/reset/endEntIter (iterators), getEntType/Topo, getEntAdj, getVtxCoord
- Arr (Entity arrays): like Entity, but for arrays of entities
- Modify: createVtx/Ent, setVtxCoord, deleteEnt

Imesh API Summary (cont.)

- From iBase:
 - Tag: create/destroyTag, getTagName/SizeBytes/SizeValues/Handle/Type
 - EntTag: get/setData, get/setInt/Dbl/EHData, getAllTags, rmvTag
 - ArrTag: like EntTag, but for arrays of entities
 - SetTag: like EntTag, but for entity sets
 - EntSet: create/destroyEntSet, add/remove entity/entities/set, isEnt/EntSetContained
 - SetRelation: add/rmvPrntChld, isChildOf, getNumChld/Prnt, getChldn/Prnts
 - SetBoolOps: subtract, intersect, unite
- iBase-inherited function names still start with 'iMesh_' to avoid name collision with other iBase-inherited interfaces (iGeom, iRel, etc.)

Slightly More Complicated Example: FindConnect (C)

```
#include <iostream>
#include "iMesh.h"

typedef void* EntityHandle;

int main( int argc, char *argv[] )
{
    // create the Mesh instance
    iMesh_Instance mesh;
    int ierr;
    iMesh_newMesh("", &mesh, &ierr, 0);

    // load the mesh
    iMesh_load(mesh, 0, "125hex.vtk", "",
               &ierr, 10, 0);

    // get all 3d elements
    iMesh_EntityHandle *ents;
    int ents_alloc = 0, ents_size;
    iMesh_getEntities(mesh, 0, iBase_REGION,
                     iMesh_ALL_TOPOLOGIES,
                     &ents, &ents_alloc,
                     &ents_size, &ierr);

    int vert_uses = 0;
```

```
    // iterate through them
    for (int i = 0; i < ents_size; i++) {
        // get connectivity
        iBase_EntityHandle *verts;
        int verts_alloc = 0, verts_size;

        iMesh_getEntAdj(mesh, ents[i], iBase_VERTEX,
                       &verts, &verts_alloc, &verts_size,
                       &ierr);

        // sum number of vertex uses
        vert_uses += verts_size;
        free(verts);

        // now get adjacencies in one big block
        iBase_EntityHandle *allv;
        int *offsets;
        int allv_alloc = 0, allv_size,
            offsets_alloc = 0, offsets_size;
        iMesh_getEntArrAdj(mesh, ents, ents_size,
                          iBase_VERTEX,
                          &allv, &allv_alloc, &allv_size,
                          &offsets, &offsets_alloc, &offsets_size,
                          &ierr);

        // compare results of two calling methods
        if (allv_size != vert_uses)
            std::cout << "Sizes didn't agree" << std::endl;
        else
            std::cout << "Sizes did agree" << std::endl;

        return true;
    }
```

FindConnect (C) Notes

- Typical inout list usage
 - `X *list, int list_alloc = 0, int list_size`
 - Setting `list_alloc` to zero *OR* `list = NULL` indicates list is unallocated, so it will be allocated inside `iMesh_getEntities`
 - Addresses of these parameters passed into `iMesh_getEntities`
- Inout list declared inside 'for' loop
- Memory de-allocated inside loop

Slightly More Complicated Example: FindConnect (Fortran)

```

program findconnect
#include "iMesh_f.h"

c declarations
iMesh_Instance mesh
integer*8 ents
pointer (rpents, ents(0:*))
integer*8 rpverts, rpallverts, ipoffsets
1 pointer (rpverts, verts(0:*))
1 pointer (rpallverts, allverts(0:*))
1 pointer (ipoffsets, ioffsets(0,*))
integer ierr, ents_alloc, ents_size
integer verts_alloc, verts_size
integer allverts_alloc, allverts_size
integer offsets_alloc, offsets_size

c create the Mesh instance
call iMesh_newMesh("MOAB", mesh, ierr)

c load the mesh
call iMesh_load(%VAL(mesh), %VAL(0),
1 "125hex.vtk", "", ierr)

c get all 3d elements
ents_alloc = 0
call iMesh_getEntities(%VAL(mesh),
1 %VAL(0), %VAL(iBase_REGION),
2 1 %VAL(iMesh_ALL_TOPOLOGIES),
1 rpents, ents_alloc, ents_size,
1 ierr)

```

```

ivert_uses = 0

c iterate through them;
do i = 0, ents_size-1
c get connectivity
verts_alloc = 0
3 call iMesh_getEntAdj(%VAL(mesh),
1 %VAL(ents(i)), %VAL(iBase_VERTEX),
1 rpverts, verts_alloc, verts_size, ierr)
c sum number of vertex uses
vert_uses = vert_uses + verts_size
call free(rpverts)
4 end do

c now get adjacencies in one big block
allverts_alloc = 0
offsets_alloc = 0
call iMesh_getEntArrAdj(%VAL(mesh),
1 %VAL(rpents), %VAL(ents_size),
1 %VAL(iBase_VERTEX), rpallverts,
1 allverts_alloc, allverts_size, ipoffsets,
1 offsets_alloc, offsets_size, ierr)

c compare results of two calling methods
if (allverts_size .ne. vert_uses) then
write(*, '("Sizes didn't agree!")')
else
write(*, '("Sizes did agree!")')
endif

end

```

FindConnect (Fortran) Notes

- Cray pointer usage
 - “pointer” (rpverts, rpoffsets, etc.) declared as type integer
 - Careful – integer*8 or integer*4, 64- or 32-bit
 - “pointee” (verts, ioffsets, etc.) implicitly typed or declared explicitly
 - pointer statement equivalences pointer to start of pointee array
 - pointee un-allocated until explicitly allocated
- Set allocated size (ents_alloc) to zero to force allocation in iMesh_getEntities; arguments passed by reference by default, use %VAL extension to pass by value; pointers passed by reference by default, like arrays
- Allocated size set to zero to force re-allocation in every iteration of do loop
- Use C-based free function to de-allocate memory

FindConnect Makefile

```
include /sandbox/tautges/MOAB/lib/iMesh-Defs.inc

FC = ${iMesh_FC}
CXX = g++
CC = gcc

CXXFLAGS = -g
CFLAGS = -g
FFLAGS = -g
FLFLAGS = -g

FindConnectC: FindConnectC.o
$(CC) $(CFLAGS) -o $@ FindConnectC.o ${IMESH_LIBS}

FindConnectF: FindConnectF.o
$(FC) -o $@ FindConnectF.o ${IMESH_LIBS}

.cpp.o:
${CXX} -c ${CXXFLAGS} ${IMESH_INCLUDES} $<
.cc.o:
${CC} -c ${CFLAGS} ${IMESH_INCLUDES} $<
.F.o:
${FC} -c ${FFLAGS} ${IMESH_INCLUDES} $<
```

ListSetsNTags Example

- Read in a mesh
- Get all sets
- For each set:
 - Get tags on the set and names of those tags
 - If tag is integer or double type, also get value
 - Print tag names & values for each set
- Various uses for sets & tags, most interesting ones involve both together
 - Geometric topology
 - Boundary conditions
 - Processor decomposition

ListSetsNTags Example (C++)

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include "iMesh.h"

#define ERRORR(a) {if (iBase_SUCCESS != err) {std::cout << a <<
    std::endl; return err;}}

int main( int argc, char *argv[] )
{
    if (argc < 2) {
        std::cout << "Usage: " << argv[0] << " <filename>" <<
            std::endl;
        return 1;
    }

    // Check command line arg
    char *filename = argv[1];

    // create the Mesh instance
    iMesh_Instance mesh;
    int err;
    iMesh_newMesh(NULL, &mesh, &err, 0);

    iBase_EntitySetHandle root_set;
    iMesh_getRootSet(mesh, &root_set, &err);
    ERRORR("Couldn't get root set.");

    // load the mesh
    iMesh_load(mesh, root_set, filename, NULL, &err,
        strlen(filename), 0);
    ERRORR("Couldn't load mesh.");

    // get all sets
    iBase_EntitySetHandle *sets = NULL;
    int sets_alloc = 0, sets_size;
    iMesh_getEntSets(mesh, root_set, 1, &sets, &sets_alloc,
        &sets_size, &err);

    // iterate through them, checking whether they have tags
    iBase_TagHandle *tags = NULL;
    int tags_alloc = 0, tags_size;
    int i, j;
```

```
    for (i = 0; i < sets_size; i++) {
        // get connectivity
        iMesh_getAllEntSetTags(mesh, sets[i], &tags, &tags_alloc,
            &tags_size, &err);

        if (0 != tags_size) {
            std::cout << "Set " << sets[i] << "; Tags:" << std::endl;

            // list tag names on this set
            for (j = 0; j < tags_size; j++) {
                char tname[128];
                int int_val, tname_size = 128;
                double dbl_val;
                iMesh_getTagName(mesh, tags[j], tname, &err, tname_size);
                tname[tname_size] = '\0';
                std::cout << tname;

                1 int tag_type;
                iMesh_getTagType(mesh, tags[j], &tag_type, &err);
                2 ERRORR("Failed to get tag type.");
                if (iBase_INTEGER == tag_type) {
                    iMesh_getEntSetIntData(mesh, sets[i], tags[j], &int_val, &err);
                    std::cout << "(val = " << int_val << "); ";
                }
                else if (iBase_DOUBLE == tag_type) {
                    iMesh_getEntSetDbldata(mesh, sets[i], tags[j], &dbl_val, &err);
                    std::cout << "(val = " << dbl_val << "); ";
                }
                else std::cout << "; ";
            }
            std::cout << std::endl;

            free(tags);
            tags = NULL;
            tags_alloc = 0;
        }

        free(sets);
        iMesh_dtor(mesh, &err);
        return 0;
    }
```

ListSetsNTags Example Notes

- Enumerated variables declared in SIDL-based code as `iface::enumNAME`, e.g. `iBase::EntityType` or `iBase::TagType`
- Enumerated variable values appear as `iface::enumNAME_enumVALUE`, e.g. `iMesh::EntityTopology_TETRAHEDRON` or `iBase::TagType_INTEGER`

Perf kernel example

Get element vertex locations, average

MOAB

```
void query_elem_to_vert()
{
    Range all_hexes;
    ErrorCode result = gMB-
>get_entities_by_type(0, MBHEX,
all_hexes);
    const EntityHandle *connect;
    int num_connect;
    double dum_coords[24];
    for (Range::iterator eit =
all_hexes.begin(); eit !=
all_hexes.end(); eit++) {
        result = gMB-
>get_connectivity(*eit, connect,
num_connect);
        assert(MB_SUCCESS == result);
        result = gMB->get_coords(connect,
num_connect, dum_coords);
        assert(MB_SUCCESS == result);

        // compute the centroid
        double centroid[3] = {0.0, 0.0,
0.0};
        for (int j = 0; j < 24;) {
            centroid[0] += dum_coords[j++];
            centroid[1] += dum_coords[j++];
            centroid[2] += dum_coords[j++];
        }
    }
}
```

iMesh

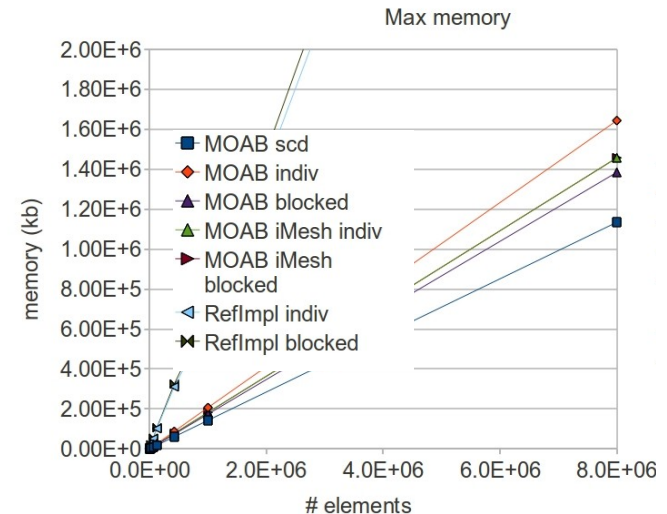
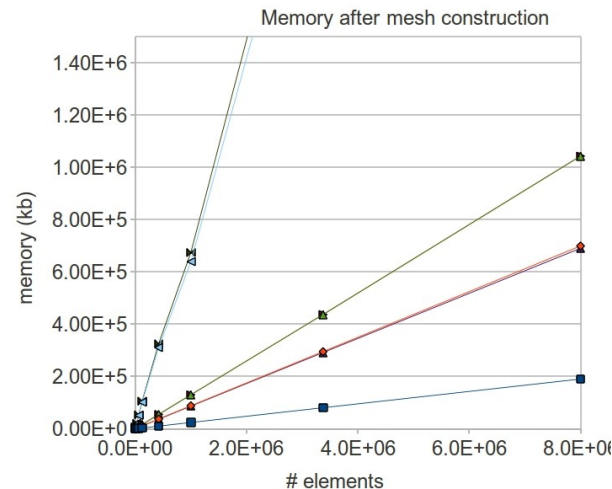
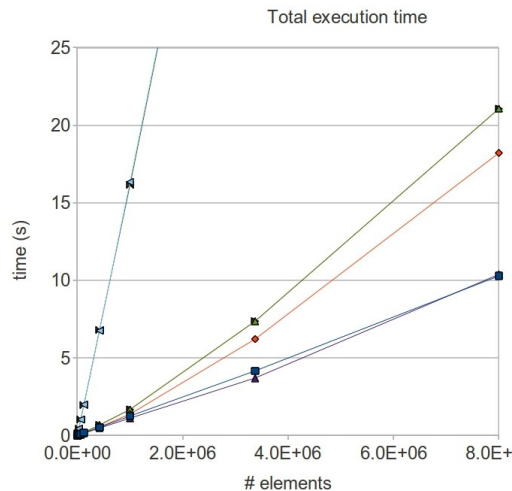
```
void query_elem_to_vert(iMesh_Instance
mesh)
{
    iBase_EntityHandle *all_hexes = NULL;
    int all_hexes_size, all_hexes_allocated
= 0;

    // get all the hex elements
    int success;
    iBase_EntitySetHandle root_set;
    iMesh_getRootSet(mesh, &root_set,
&success);
    iMesh_getEntities(mesh, root_set,
iBase_REGION, iMesh_HEXAHEDRON,
        &all_hexes, &all_hexes_allocated,
&all_hexes_size, &success);
    // now loop over elements
    iBase_EntityHandle *dum_connect = NULL;
    int dum_connect_allocated = 0,
dum_connect_size;
    double *dum_coords = NULL;
    int dum_coords_size,
dum_coords_allocated = 0;
    int order;
    iMesh_getDfltStorage(mesh, &order,
&success);

    for (int i = 0; i < all_hexes_size; i++)
```

Performance

- Large applications balance memory and cpu time performance
- Implementations of iMesh vary on speed vs. memory performance
 - Create, v-E, E-v query, square all-hex mesh
 - Test entity- vs. array-based access
- Compare MOAB Native/structured/iMesh, RefImpl iMesh



iMesh Review

- Data model consists of 4 basic types: Interface, Entity, Entity Set, Tag
- Applications reference instances of these using opaque handles
- ITAPS interfaces use C-based APIs, for efficiency and interoperability
- Not covered here:
 - Iterators (intermediate-level, “chunked” access to mesh)
 - Modify (relatively coarse-grained, basically create and delete whole entities)
 - Set parent-child links

MOAB

- MOAB entity types include entity set & are sorted by dimension (defined in EntityType.h)
 - MBVERTEX, MBEDGE, MBTRI, MBQUAD, MBPOLYGON, MBTET, ..., MBENTITYSET
- EntityHandle properties
 - MOAB entity handle is an integer type, embeds entity type, entity id
 - List of contiguous handles can be stored in ranges, const-space lists
 - Sort by type, dimension
 - Set booleans (intersect, union, subtract) very fast on ranges
 - Config option to use 64-bit handles on 32-bit apps if id space is a concern
- Range class: series of sub-ranges of entity handles
- MOAB functionality accessed through Interface, an abstract base class
 - Most functionality similar to what's in iMesh, plus a little more

MOAB Tools

- Some functions support set booleans implicitly:

```
virtual MBERrorCode get_adjacencies(const MBRange &from_entities,
                                   const int to_dimension, const bool create_if_missing,
                                   MBRange &adj_entities,
                                   const int operation_type = MBInterface::INTERSECT);
```

- For multiple from_entities, adj_entities will be intersection of queries on each from_entity
 - To get common vertices between two entities, call with to_dimension=0 and MBInterface::INTERSECT
 - To get all vertices used by group of entities, call with to_dimension=0 and MBInterface::UNION
- MBSkinner: gets skin (bounding (d-1)-dimensional entities) of a set of entities
- IO: format designated by file extension;
 - CUBIT .cub (R), Exodus (.g, .exoll) (RW), vtk (.vtk) (RW), native HDF5 format (.h5m, .mhdf) (RW)
 - Use .h5m/.mhdf to save everything MOAB can represent in data model (sets, tags, set parents/children, polygons/polyhedra, etc.)

MOAB Parallel

- Configure with `--with-mpi=` option
- Parallel read (`bcast_delete`, `read_delete`) working
 - Can use any “covering” set of sets as partition
 - Read method designated with options to MOAB's `load_file` function, e.g.
`“PARALLEL=BCAST_DELETE;PARALLEL_PARTITION=MATERIAL_SET”`
`“PARALLEL=BCAST_DELETE;PARALLEL_PARTITION=GEOM_DIMENSION;PARTITION_VAL=3;PARTITION_DISTRIBUTE”`
- Other classes
 - `ParallelComm`: pass entities/tags/sets between processors, define communicator
 - `ParallelData`: convenience functions for getting partition, interface entities
- Relevant tags (defined in `MBParallelConventions.h`):
 - `PARALLEL_SHARED_PROC`: 2 ints, ranks of sharing procs on 2-proc interface
 - `PARALLEL_SHARED_PROCS`: N ints, ranks of sharing procs when > 2 procs share iface
 - `PARALLEL_OWNER`: rank of owning processor for interface entities, sets
 - `PARALLEL_GHOST`: rank of owning processor for ghost entities, sets
 - `PARALLEL_GID`: global id, used to match vertices, other entities

MOAB Parallel Example

(condensed from mbparallelcomm_test.cpp in MOAB source dir)

```
int main(int argc, char **argv)
{
    int err = MPI_Init(&argc, &argv);
    int nprocs, rank;
    err = MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    err = MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // create MOAB instance based on that
    MBInterface *mbImpl = new MBCore(rank, nprocs);
    MBParallelComm *pcomm = new MBParallelComm(mbImpl);

    // read a file in parallel
    const char *options =
        "PARALLEL=BCAST_DELETE;PARTITION=GEOM_DIMENSION;PARTITION_VAL=3;PARTITION_DISTRIBUTE";
    MBEntityHandle file_set;
    MBErrorCode result = mbImpl->load_file(filename, file_set, options);

    // resolve shared vertices
    result = pcomm->resolve_shared_ents();

    // get shared vertices on this proc
    MBRange shared_ents;
    result = pcomm->get_shared_entities(0, shared_ents);

    MPI_Finalize();
    return 0;
}
```

ITAPS Interfaces Best Practices

- Pre-allocate memory in application or re-use memory allocated by implementation
 - E.g. getting vertices adjacent to element – can use static array, or application-native storage
- Take advantage of implementation-provided capabilities to avoid re-inventing
 - Partitioning, IO, parallel communication, (parallel) file readers
- Be careful about integer*8, integer*4, memory corruption in Fortran apps (valgrind is your friend)
- Implement iMesh on top of your data structure
 - Take advantage of tools which work on iMesh API
- Let us help you
 - Not all the tricks can be easily described and may not be self-evident

Best Practices: MOAB vs. ITAPS

- When handling & manipulating large lists of entity handles, using MBRange can save lots of memory
 - But small or fragmented ranges can cost in terms of time
- Lower overhead with MOAB native interface
 - Actual amount depends on granularity of access
- No MOAB Fortran or C interface, only C++

MOAB Code Details

- Build/configure: autotools (including libtool)
 - Contributed support for cmake
- Compilers: GNU (including gfortran), IBM XL, Intel, others
- Platform: Linux, AIX, Mac OS X
 - Windows: maybe eventually, but no strong (paying customer) pull
 - Parallel: IBM BG/P, clusters (Jaguar probably not difficult)
- Open source (LGPL) throughout
- Subversion, world-readable repository
- Language:
 - C++ throughout (iMesh API is C)
 - Namespace-protected by default, but non-namespace for legacy apps
- I/O
 - Native: HDF5-based
 - Vtk, ExodusII, partial Netcdf nc, others
- 32, 64 bit both supported

Recent improvements/plans

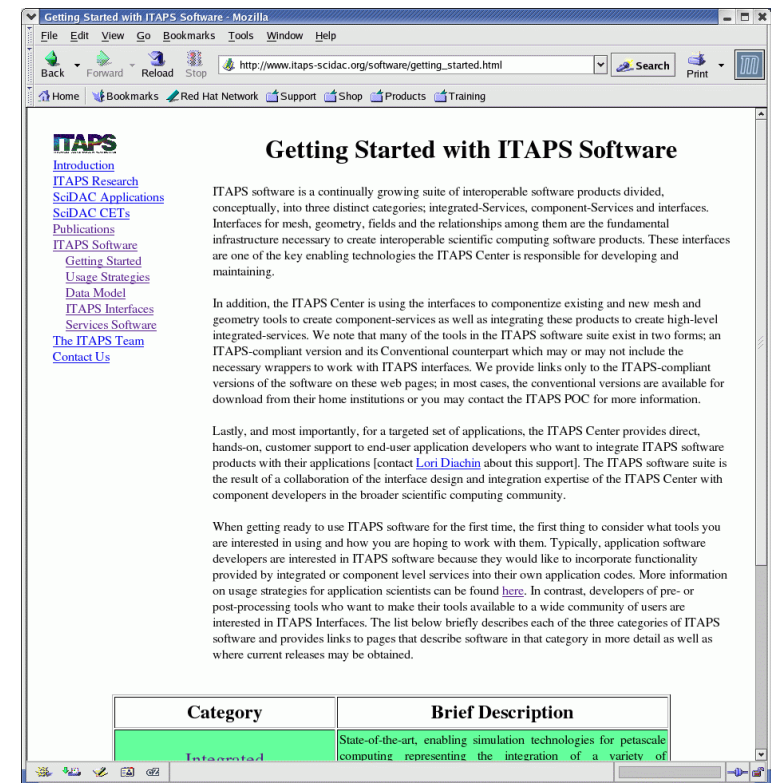
- Started nc data reader (just vertices for now, but whole mesh, fields very soon)
- Direct access to tag data storage
 - For static-mesh applications, eliminates cost of going through API, could be significant savings for fine-grained access
- Solution transfer between meshes
 - Current interpolation-based capability inside MOAB proper
 - Parallel-aware already
 - 3D only, tet, hex FE shape functions
 - Will develop more fully using Intrepid as part of this project
 - Basis for solution coupler development under CSSEF, so more to come
- Time-dependent data
 - Currently, have to “fake it” with different tag for each time step
 - Will develop native support for time-dependent tags

Obtaining the ITAPS Software

ITAPS Software Web Pages

<http://www.itaps.org/software/>

- Provides help getting started
- Usage strategies
- Data model description
- Access to interface specifications, documentation, implementations
- Access to compatible services software



Getting Started with ITAPS Software

ITAPS software is a continually growing suite of interoperable software products divided, conceptually, into three distinct categories; integrated-Services, component-Services and interfaces. Interfaces for mesh, geometry, fields and the relationships among them are the fundamental infrastructure necessary to create interoperable scientific computing software products. These interfaces are one of the key enabling technologies the ITAPS Center is responsible for developing and maintaining.

In addition, the ITAPS Center is using the interfaces to componentize existing and new mesh and geometry tools to create component-services as well as integrating these products to create high-level integrated-services. We note that many of the tools in the ITAPS software suite exist in two forms; an ITAPS-compliant version and its Conventional counterpart which may or may not include the necessary wrappers to work with ITAPS interfaces. We provide links only to the ITAPS-compliant versions of the software on these web pages; in most cases, the conventional versions are available for download from their home institutions or you may contact the ITAPS POC for more information.

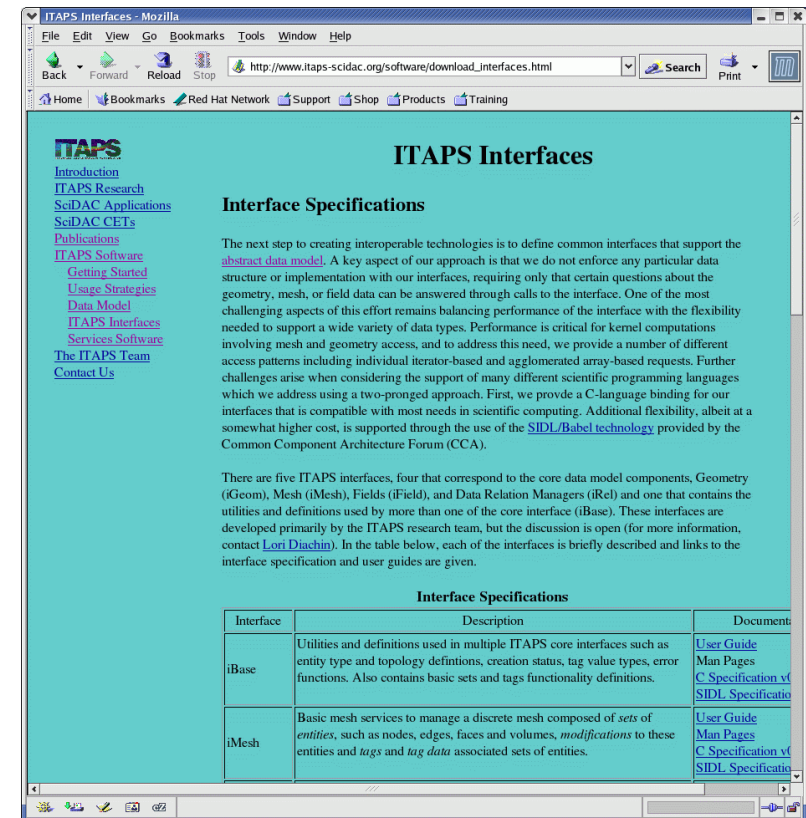
Lastly, and most importantly, for a targeted set of applications, the ITAPS Center provides direct, hands-on, customer support to end-user application developers who want to integrate ITAPS software products with their applications [contact [Lori Diachin](#) about this support]. The ITAPS software suite is the result of a collaboration of the interface design and integration expertise of the ITAPS Center with component developers in the broader scientific computing community.

When getting ready to use ITAPS software for the first time, the first thing to consider what tools you are interested in using and how you are hoping to work with them. Typically, application software developers are interested in ITAPS software because they would like to incorporate functionality provided by integrated or component level services into their own application codes. More information on usage strategies for application scientists can be found [here](#). In contrast, developers of pre- or post-processing tools who want to make their tools available to a wide community of users are interested in ITAPS Interfaces. The list below briefly describes each of the three categories of ITAPS software and provides links to pages that describe software in that category in more detail as well as where current releases may be obtained.

Category	Brief Description
Integrated	State-of-the-art, enabling simulation technologies for petascale computing representing the integration of a variety of

Interface Software Access

- Links to the interface user guides and man pages where available
- Links to the interface docs and headers
- Links to implementations for iMesh, iGeom, iRel
 - Links to the home pages for more information
- Simple examples, compliance testing tools and build skeletons coming soon



ITAPS Interfaces

Interface Specifications

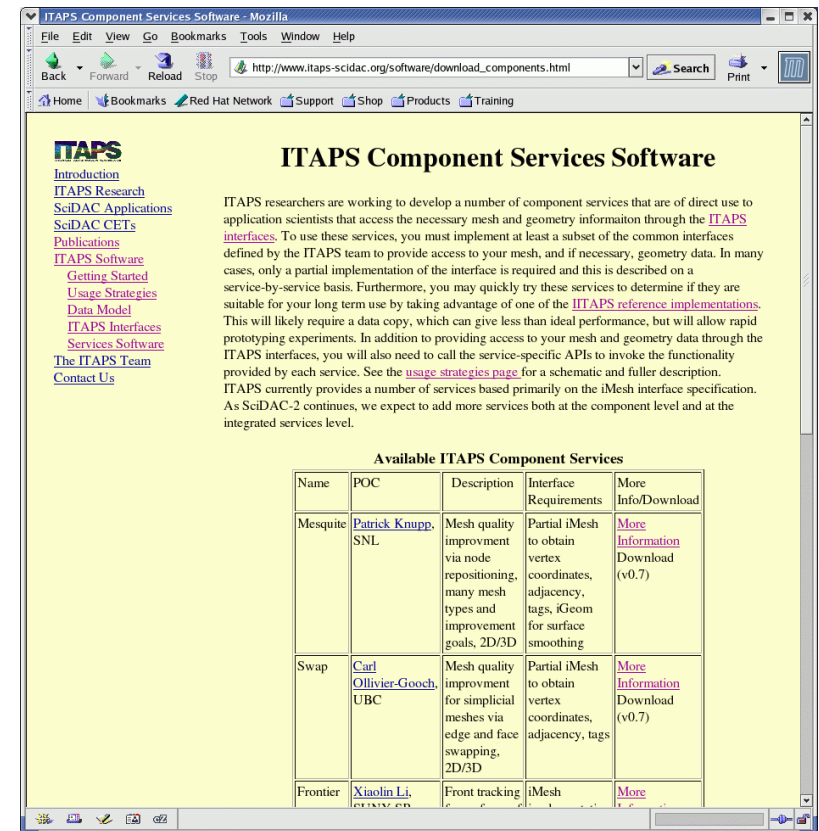
The next step in creating interoperable technologies is to define common interfaces that support the [abstract data model](#). A key aspect of our approach is that we do not enforce any particular data structure or implementation with our interfaces, requiring only that certain questions about the geometry, mesh, or field data can be answered through calls to the interface. One of the most challenging aspects of this effort remains balancing performance of the interface with the flexibility needed to support a wide variety of data types. Performance is critical for kernel computations involving mesh and geometry access, and to address this need, we provide a number of different access patterns including individual iterator-based and agglomerated array-based requests. Further challenges arise when considering the support of many different scientific programming languages which we address using a two-pronged approach. First, we provide a C-language binding for our interfaces that is compatible with most needs in scientific computing. Additional flexibility, albeit at a somewhat higher cost, is supported through the use of the [SIDL/Babel technology](#) provided by the Common Component Architecture Forum (CCA).

There are five ITAPS interfaces, four that correspond to the core data model components, Geometry (iGeom), Mesh (iMesh), Fields (iField), and Data Relation Managers (iRel) and one that contains the utilities and definitions used by more than one of the core interface (iBase). These interfaces are developed primarily by the ITAPS research team, but the discussion is open (for more information, contact [Lori Diachin](#)). In the table below, each of the interfaces is briefly described and links to the interface specification and user guides are given.

Interface	Description	Document
iBase	Utilities and definitions used in multiple ITAPS core interfaces such as entity type and topology definitions, creation status, tag value types, error functions. Also contains basic sets and tags functionality definitions.	User Guide Man Pages C Specification v1 SIDL Specification
iMesh	Basic mesh services to manage a discrete mesh composed of sets of entities, such as nodes, edges, faces and volumes, modifications to these entities and tags and tag data associated sets of entities.	User Guide Man Pages C Specification v1 SIDL Specification

Services Software Access

- Links to the services built on the ITAPS interfaces
- Currently or very soon to be available
 - Mesquite
 - Zoltan
 - Swapping
 - Frontier
 - VisIt Plug In
- Links to home pages for more information
- Instructions for build and links to supporting software



The screenshot shows a Mozilla browser window titled "ITAPS Component Services Software - Mozilla". The address bar shows the URL "http://www.itaps-scidac.org/software/download_components.html". The page content includes the ITAPS logo, a navigation menu with links like "Introduction", "ITAPS Research", "SciDAC Applications", "SciDAC CETs", "Publications", "ITAPS Software", "Getting Started", "Usage Strategies", "Data Model", "ITAPS Interfaces", "Services Software", "The ITAPS Team", and "Contact Us". The main heading is "ITAPS Component Services Software". Below this, a paragraph explains that ITAPS researchers are developing component services for application scientists. A table titled "Available ITAPS Component Services" lists three services: Mesquite, Swap, and Frontier. Each row includes the service name, a POC (Point of Contact), a description of the service, the interface requirements, and a link to more information or download.

Name	POC	Description	Interface Requirements	More Info/Download
Mesquite	Patrick Knupp, SNL	Mesh quality improvement via node repositioning, many mesh types and improvement goals, 2D/3D	Partial iMesh to obtain vertex coordinates, adjacency, tags, iGeom for surface smoothing	More Information Download (v0.7)
Swap	Carl Ollivier-Gooch, UBC	Mesh quality improvement for simplicial meshes via edge and face swapping, 2D/3D	Partial iMesh to obtain vertex coordinates, adjacency, tags	More Information Download (v0.7)
Frontier	Xiaolin Li, CERN, CO	Front tracking	iMesh	More Information Download (v0.7)

MOAB Software Web Pages

<http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB>

- General information
- Browse svn repo
- FAQ
- Pointers to mailing list archives